Modular Analytics Framework for Rapid AI/ML Training and Deployment

Ronald Deiotte
ISSAC LLC
Colorado Springs, CO
ron.deiotte@issaccorp.com

Lauren Britton
ISSAC LLC
Huntsville, AL
lauren.britton@issaccorp.com

ABSTRACT

Contrary to common belief, most of data science is focused on the gathering and preparation of data to be fed into analytic algorithms and then interpreting the results, rather than the actual development of those algorithms. We often break down the process of data science into 5 steps: 1) identify a problem, 2) gather the data, 3) clean the data, 4) analyze the data, and 5) present the result. While these 5 steps seem nice and clean, the reality is that data scientists often need to be working on steps 3, 4, and 5 before they even have the data, or in the case that a new data source is discovered that could refine existing analysis. This more chaotic environment means that code and processes developed to solve one identified problem can only be used on that problem and cannot be used for other problems. This is often because steps 2-5 are combined into a monolithic process where data is explicitly passed from data gathering processes directly into data cleaning processes and so on.

A Modular Analytics Framework (MAF) will separate the data passing infrastructure from the data manipulation processes. This allows for the clean definition of the processes as modules that can be added, changed, and connected at will within a data infrastructure. In this paper, we discuss the requirements for a MAF and introduce a standardized definition for describing data, modules, and pipelines. Finally, we explore some more complicated example uses of a MAF including competing Machine Learning (ML) configurations and techniques, and continuous training and deployment of Artificial Intelligence (AI) models.

ABOUT THE AUTHORS

Ronald Deiotte is the chief scientist at ISSAC. Ronald has been developing data analytics, data management systems, and novel databases for flexible application to M&S, data analysis, and model-based systems engineering with ISSAC for 15 years. In that time, he has worked projects that span from missile defense to cancer research to overland shipping logistics. This multi-disciplinary environment has led him to several new approaches and unique solutions to problems that span all data environments.

Lauren Britton is a Data Scientist at ISSAC LLC. She holds a BS in Mathematics and an MS in Business Analytics, both from the University of Alabama in Huntsville. Most of Lauren's career has been focused on providing systems engineering, technical assistance, and data analytics support to the Army. Her interests include predictive modeling and advanced data analytics, specifically the extract, transform, load, and data visualization processes.

Modular Analytics Framework for Rapid AI/ML Training and Deployment

Ronald Deiotte
ISSAC LLC
Colorado Springs, CO
ron.deiotte@issaccorp.com

Lauren Britton
ISSAC LLC
Huntsville, AL
lauren.britton@issaccorp.com

INTRODUCTION

Data scientists are often tasked with identifying what information, knowledge, or conclusions can be discovered from a data set (Jamshidi, 2008). In this process, data scientists often use a well-worn set of tools: basic statistics, trend lines, etc. These tools are readily available and easy to incorporate into most environments. In some cases, it is possible to use more advanced tools that might fall into AI or ML like neural nets, adversarial networks, or predictive algorithms. However, there are several aspects of the modern data environment that make the use of such tools difficult. The fact that many data sources are often large scale, distributed, and secured means that the data scientist must be a multi-disciplinary expert. They must know how to handle large scale data, they must be a cloud expert to access distributed data, and they must be a cyber security expert to safely access and maintain data security of both the original data and of the derived or extracted data that they are interested in. A data scientist must do all of this within the context of knowledge discovery where they may not know exactly what data they will need or what is available. The constraints that arise from this often lead to scientists being limited to just the simpler statistics and trend line type analysis for knowledge discovery. Because of this, data science processing involving AI and ML are only introduced later in development, if at all.

A way to allow for more rapid data access and tool deployment is to use a Modular Analytics Framework (MAF). A MAF is a set of tools, services, and data infrastructure that allows users to rapidly develop, test, and deploy a data science workflow in a digital environment. This is done by separating the functionality of the data science workflow into infrastructure and modules. In a typical data science workflow, data is brought into the system, analysis is done on the data, intermediate results are stored and potentially analyzed further, and then the findings and products are reported to the analyst or deployed in other settings. In a MAF, the data handling is considered part of the infrastructure of the MAF, and the analysis, deployment, and reporting aspects are handled by configurable low-code generic analytic modules.

Separating the data handling from the execution of the analysis has several benefits that will be discussed in detail in this text, but one high level benefit is the alignment between MAFs and well-known Model Based Systems Engineering (MBSE) documents like Class Diagrams and Block Diagrams (OMG Systems Modeling Language (OMG SysML™) Version 1.6, 2019). These MBSE documents can effectively be thought of as templates for the MAF modules on the data infrastructure. The Block Diagram is perhaps the better fit because it is possible to make a Block Diagram at high levels of abstraction. A MAF provides a powerful tool for moving from design and development to an actual data science product. This is particularly true when designing for AI/ML processes and life cycles where there are existing tools and approaches that are in common use but that cannot stand alone and need some development to be effective.

MODULAR ANALYTICS IN PRACTICE

For most data science exercises, the entire process is either contained within a single monolithic codebase – making it hard to update and maintain – or much of the data handling depends on human-in-the-loop – slowing the process and introducing the possibility of human error. A MAF avoids these problems by separating out logically distinct algorithms and packaging them into small, easy to test, replaceable modules. However, modules do need to have the correct configuration parameters exposed. These configuration parameters are often set within the monolithic code and not exposed for general use. Further, a MAF needs to have built-in tooling to ensure that each algorithm is receiving the data required for the implemented algorithm. This is similar to the difference between sequential programming and object-oriented programming. In sequential programming, typing is simple because everything is defined within the same code stack; in object-oriented programming, objects can be injected from outside the code stack so typing and clearly defining objects becomes much more important. While keeping track of objects and types in object-oriented programming is more difficult, it is a much more flexible, maintainable, and updatable approach to programming. With MAF, we must publish information about each module and the expected data inputs and outputs, but we gain the ability to quickly change out and deploy tools, updating or expanding a data science workflow.

There are several goals of a MAF. The first goal is to encourage code reuse and allow for low code implementations. This is important for both the lifecycle of the individual modules and for the lifecycle of the workflows that are built from those modules. The next main goal is to allow modules to be easily replaced and run in parallel. This is particularly important when doing data exploration and knowledge discovery. Next, a MAF should be able to handle a high throughput of data allowing for analysis of large data sets and the use of advanced analytics on those large data sets. The last major goal of a MAF is to normalize the deployment environment. This means that the use of the MAF should not depend on the backend environment in which it is deployed. In other words, a MAF should work the same, from the perspective of the user, if it is deployed on a personal laptop or into Azure or Amazon Web Services (AWS).

Existing MAF-Like Products

As the data set becomes more complex, which tends to be the case for deep learning applications, data scientists may require more flexibility than what is provided in traditional data analytics tools. Recognizing this need, some platforms like Tableau attempt to provide more flexibility in their architecture but only focus on a subset of the entire data pipeline. Popular end-to-end platforms like Alteryx and KNIME have adopted a modular approach to analytics, allowing users to better tailor data workflows to their specific needs. Users can create their own tools, or they may select from an extensive list of pre-built tools, ranging from data ingestion to report writing. Additionally, both platforms provide both built-in machine learning tools and support integration with external machine learning libraries (Platform Analytics Capabilities | Alteryx_2024, 2024) (Software overview | KNIME, 2024).

These tools are known for their user-friendly environments and usefulness in making data accessible to all data users. Some data scientists have even seen a rise in productivity when adopting KNIME when compared to using computation notebooks. According to Einblick Analytics, the most significant increase in productivity was when using data boards, their novel concept for collaborative data analysis, when compared to using computational notebooks. Data boards build upon the features in commercial workflow automation tools and combine them with those of Google Docs, computational notebooks, and dashboarding platforms, allowing users to collaborate throughout the entire data pipeline (Buratti, et al., 2023).

Despite their popularity, these workflow automation tools still have limitations that leave the data science community searching for alternatives. Extending the modularity concept to analytic algorithms, Hahmann et al. created an alternative to KNIME's limitations in customization of analytic algorithms by deconstructing them into small, reusable building blocks. In addition, the concept of base templates is introduced. Base templates are predefined structures that outline the phases that define analytic workflows. The base templates provide the structure of the workflow, while the building blocks are the tools or tasks to be performed at each phase of the workflow. Not only can the building blocks be combined and exchanged, so can the phase to create various other algorithms, increasing reusability and adaptation (Hahmann, Hartmann, Kegel, Habich, & Lehner, 2016).

The proliferation and existence of these tools shows that the basic concept of a MAF is now part of the common data science ecosystem. However, there are still some potential areas of expansion for the MAF concept. These areas of expansion include: an open architecture so that the MAF can be tailored to the data environment and the computational constraints of a particular domain; a more flexible module implementation to allow scientists and developers to use the best tools for the job when it comes to module implementation; the ability to take analysis to the data rather than always moving large data to a different location for analysis; and the ability to embed a MAF in a quarantined data environment where users can run workflows without having any direct access to the data. All these areas of expansion, and many more, require a generalized standard for describing data, modules, and workflows.

MAF Use-cases

There are several use-cases where a MAF might be used, some of these use-cases are currently underrepresented by existing tools (see Figure 1). It is important to explore these use-cases to better understand the eventual requirements and artifacts needed for a MAF. We first must look at the actors expected to interact with a MAF. The actor that will interact the most with a MAF is the data scientist. They are expected to be connecting data to the MAF, setting up workflows from a set of modules, exploring the library of modules, and developing new modules. In the process of setting up a workflow, an actor can reasonably be expected to deploy modules for data access, data manipulation/normalization, data analysis, product deployment, and reporting. In this case, data access modules are expected to retrieve data from an outside source and inject it into the MAF data infrastructure. Data manipulation/normalization modules perform basic operations to get data objects into a usable form for other modules. Data analysis modules perform the actual analysis and may produce findings or products, such as ML models, that can perform specific analysis tasks on the data. Finally, reporting modules allow other users to view findings and effectively publish the findings to an accessible location.

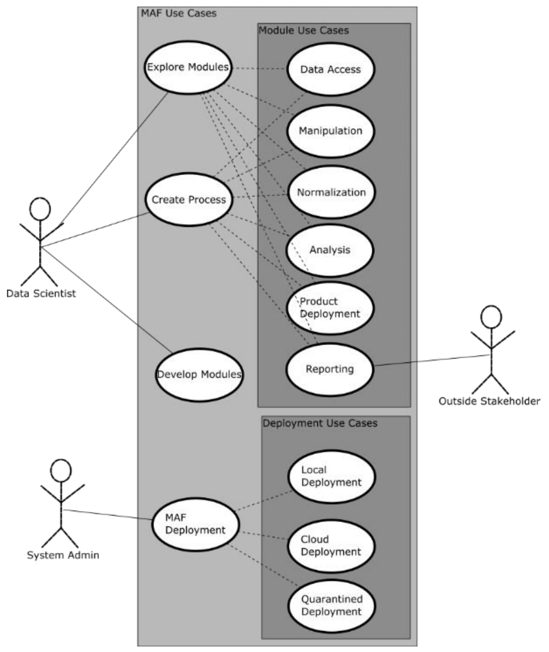

Figure 1: MAF use-cases and actors

We also must consider the system administrator that is tasked with deploying the MAF to fit the needs of the data scientist. This means accounting for the data and computational environment. In terms of data environment, this means accounting for the scale of the data as well as the access level of the data. A MAF can be deployed in any location and accessed remotely by data scientists to deploy workflows. This means that, if the MAF is expected to access a large-scale data set where the transmission of the data might be a limitation to the usefulness of the data, the MAF should be deployed on the same system that houses the data, minimizing any network issues that might arise. Similarly, data access limitations may also need to be accounted for (Framework for Access to All of Us Data Resources v1.1, 2021). Most data access limitations can be grouped into one of four categories: 1) public, where data is free to be copied and hosted by other entities; 2) centralized, where there is some desire to maintain an authoritative source, but the data can be seen by anybody; 3) restricted, where data authority is maintained and a controlled group of users is allowed to access the data; and 4) locked, where the data can only be accessed by a select few who are not, necessarily, analysts (see Table 1). If the data is locked and, in some cases, restricted, then a possible solution is to deploy a quarantined MAF on the data system. A quarantined MAF would consist of an open sandbox MAF where data scientists could develop modules and workflows using synthetic data and a remote MAF that is collocated with the real data. Once a workflow has passed some pre-defined checks in the sandbox MAF, it can be submitted to the remote MAF and run on the

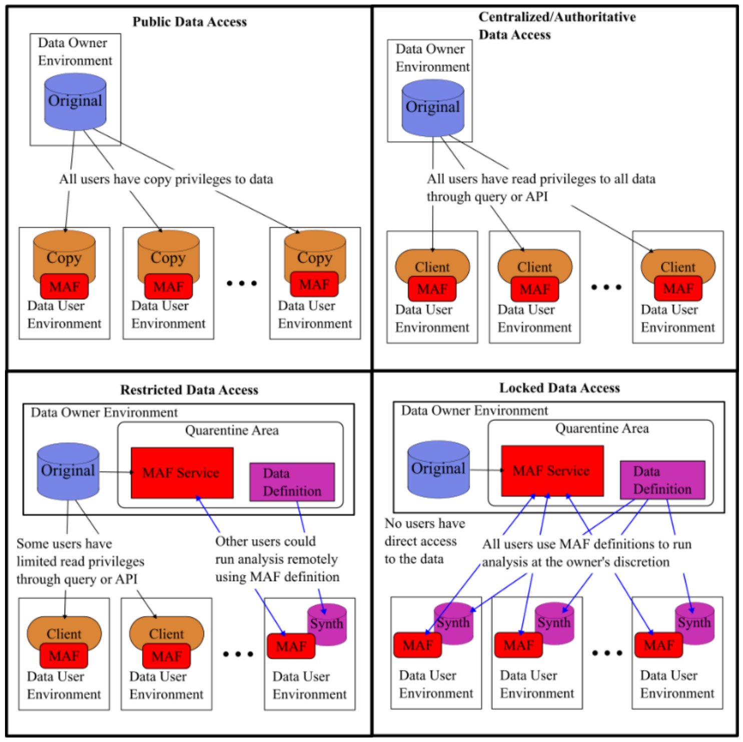
Public Data Access
Data Owner Environment
Original
All users have copy privileges to data
Copy
MAF
Data User Environment
Centralized/Authoritative Data Access
All users have read privileges to all data through query or API
Client
Restricted Data Access
Quarantine Area
MAF Service
Data Definition
Some users have limited read privileges through query or API
Other users could run analysis remotely using MAF definition
Synth
Locked Data Access
No users have direct access to the data
All users use MAF definitions to run analysis at the owner's discretion

Figure 2: Data Access Categories and MAF

real data. The results of this run would be delivered to the original data scientist only after review by the data owners. This arrangement would ensure that data is not accidently leaked outside of the controlled environment while still allowing valid use and analysis by “untrusted” users. The specifics of how to implement these different deployments are beyond the scope of this document but it is important to know that these deployments exist so we can properly define the metadata needed to make a MAF function within all these constraints.

Table 1: Deployment solutions for data scale and access level

MAF Deployment Use Cases		**Data Scale**		
		< 10 GB	**10GB to 10 TB**	**>10 TB**
Data Access Level	**1. Public**	Local	Local	Cloud/Remote
	2. Centralized	Local	Cloud/Remote	Cloud/Remote
	3. Restricted	Local (with Data Access)	Quarantined	Quarantined
	4. Locked	Quarantined	Quarantined	Quarantined

MAF WORKFLOW METADATA DEFINITIONS

With the goal of establishing an open set of rules for MAF operations, there is a need to clearly define the minimal set of information needed to set up and run an analytics workflow within a MAF. With such a definition, users could easily bring their own modules as well as use best-of-breed algorithms in their data science workflows. A clear definition for workflows, modules, and data is also essential for the quarantined data use-case. This is because it is much easier to pass small metadata files between the client and the host quarantine service rather than the full workflow or all the modules. Upon receiving a workflow definition, the quarantined service could pull needed modules from an authoritative module repository ensuring that the analysis is above board, and no malicious or unapproved modules have been introduced to the quarantined environment.

There are three types of information needed for a MAF workflow: data definitions, module definitions, and workflow definitions (Figure 3).

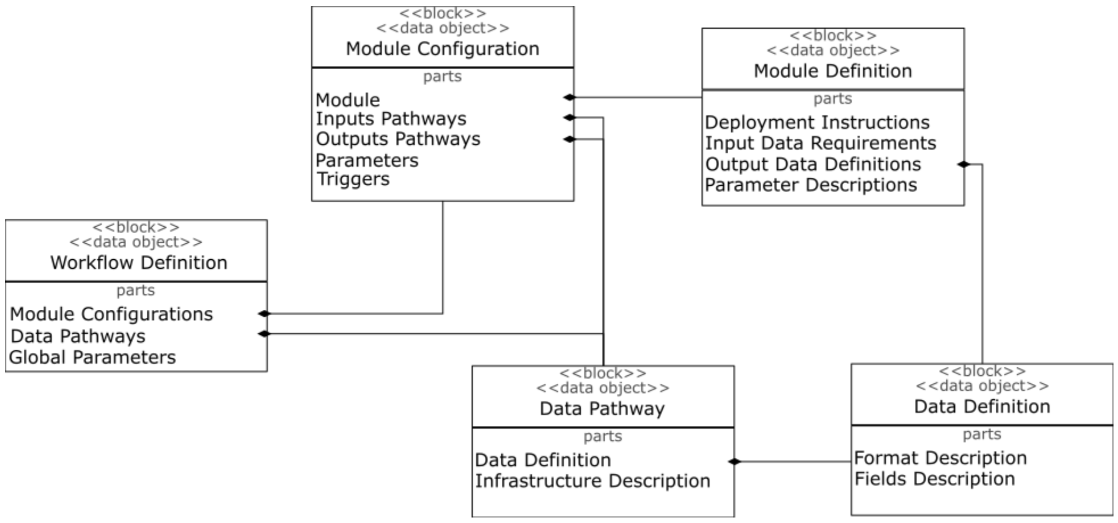

Figure 33: High level relationships between workflow metadata definitions

Data Definitions

Data definitions describe the data that is being passed between the modules of the workflow. This information allows the MAF to appropriately set up the data infrastructure (regardless of how the infrastructure is implemented). It also allows the MAF to check to make sure that modules are connected to appropriate data pathways prior to trying to execute the whole workflow, this massively reduces the chances of a module failing due to input errors. A data definition consists of the format the data is transmitted in and details about the content of an individual data object, this includes data fields with types and constraints.

Module Definitions

Module definitions describe the way a module functions. This information is used by the MAF to ensure that it is connected to the correct data pathways and to stand up an instance of the module to process data. Module definitions consist of information like source location (repository, local storage, etc.); the required infrastructure needed to run the module, this includes the execution resources (Java, Python, R, Docker, etc.) and supporting libraries; the available configuration fields for the module; and finally, the information for the expected data inputs and products for the module.

Workflow Definitions

Workflow definitions describe the overall workflow that will be run on the MAF. This information tells the MAF what modules to use, their start triggers, and how they are connected to each other. The MAF is expected to derive the data infrastructure needed based on the modules that are connected and the data that is being passed in those connections. The workflow definition also contains the configuration information for each of the modules. To simplify this part of the definition, it is expected that the workflow definition will have global configuration fields that can be mapped to individual modules. If a configuration field (global or specific) is annotated as parallel and given a list or range of values, then the MAF will instantiate several instances of the given modules. Each instantiated module will be deployed with different configuration values based on the list or range.

The full description of these definitions is beyond the scope of this paper, but we are currently in the process of refining and finalizing a full description of these definitions as a potential standard for MAF descriptions.

EXAMPLE AND EXPERIMENT

To provide an example of how a MAF can operate and to prove the ideas discussed in this document, we put together a simple sample problem. In this problem, we seek to perform a common ML task of image classification. In the past, we have used some simple ML techniques to do image recognition. When using these ML techniques for such a task, there are several variables that can be modified to increase the accuracy of the trained classifier. For this example, we are going to use a Stochastic Gradient Decent (SGD) technique to create a model to recognize images with a specific label. With an SGD, there are several parameters we can vary to get different models. How each of these parameters will affect the quality of the final model is not intuitive. In order to optimize our model, we will focus on the starting conditions of the classifier and the maximum iterations in the hope of being able to train with fewer iterations. We used a MAF to train several SGD models in parallel on the same training and testing data sets; then, we compared those trained models on how they perform on a competition data set; and finally, the MAF deployed the winning model on a simulated live data stream.

This experiment is similar to past projects that used monolithic coding practices. For this effort, we mainly focused on the deployment of a MAF workflow using the Workflow, Module, and Data Definitions that we are developing as a possible standard for future MAF descriptions. As such we are using our own implementation of a MAF that is being built to handle these definitions and as a testbed for more generalized MAF concepts. The MAF primarily consists of a module library and a workflow runner. The module library consists of a graph database containing all the metadata for the modules and a code repository that can hold standalone scripts as well as compiled and containerized code. We use a graph database for the metadata because it allows us to quickly check the data compatibility of modules as we put together a workflow.

The workflow runner is a set of processes that set up either an MQTT messaging service or a PostgreSQL instance to act as the data pathways for the workflow (MQTT: The Standard for IoT Messaging, 2024) (PostgreSQL: The World's Most Advvanced Open Source Relational Database, 2024). In both cases the workflow runner automatically adds in the connection operations to convert from the format on the data pathways to the format requested by the modules. The workflow runner also manages the running of the modules and can run bash scripts, Python scripts, and Docker containers.

Data Description

For this example, we used data from the Open Images data set. Open Images has over 9 million labeled images, with 1.9 million of them having more detailed annotations such as bounding boxes, object segmentations, and visual relationships. It also includes thousands of object classes, making it a great source for a diverse and comprehensive visual data set for training machine learning algorithms (Open Images Dataset V7: Download, 2024).

In order to train a classifier model, we used the "aircraft" class due to its relevance in defense applications, specifically in recognizing friendly and enemy aircraft in various contexts and environments. Identifying a specific object in a diverse set of images is demonstrative of broader data science problems where a model needs to generalize across different contexts and variations. This challenge is representative of many real-world problems and has multiple known solutions that perform well. Therefore, our goal is not to develop a new or better solution, but to enhance the efficiency of existing solutions by implementing our proposed MAF. The intent is to provide greater flexibility in the model's functionality while establishing universal standards that eliminate the human guesswork involved when setting up these types of problems.

MAF Workflow Description

The example workflow is done in five stages (see Figure 4): the data access stage, the data normalization stage, the training stage, the competition stage, and the deployment stage.

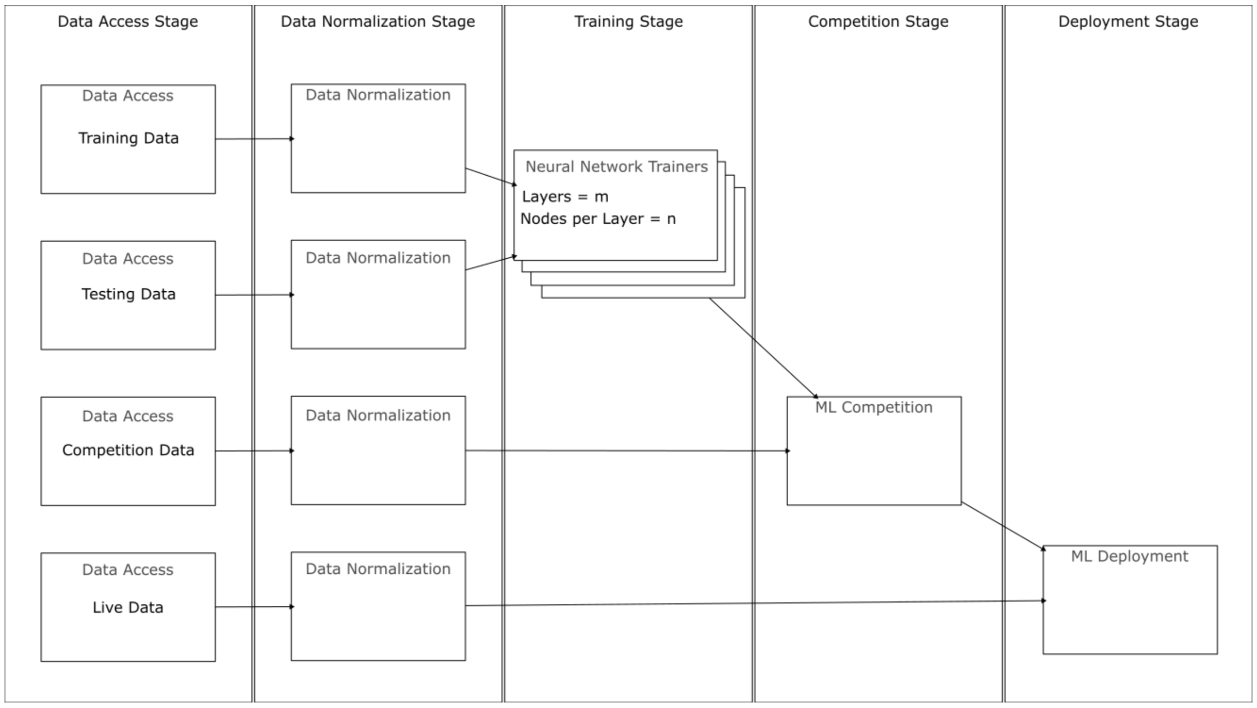

Figure 44: Example competitive AI training workflow

Data Access Stage
The data access stage uses four instances of a custom Python script to retrieve four different sets of images and labels from Open Images. These four sets are the training set, the testing set, the competition set, and the live data set. The MAF creates four different pathways in the data infrastructure for these sets. The module has several files indicating which images are in each different set. The exact file each of the four modules uses is configurable and set in the workflow definition.

Data Normalization Stage

The labels as retrieved from Open Images are in MID format meaning that they are not human readable. As we are only interested in a classifier that can identify "aircraft," we convert all the labels to a Boolean, true if it is "aircraft," false if it is not. This will also help make our resulting SGD model very targeted and more accurate. The normalized data from each of the original four pathways is put on to four more pathways created by the MAF. This normalization module is also a simple Python script.

We also normalize the images at this stage by scaling them to the same size and running the same data reduction transformations to feed simplified versions of the original images to the SGD trainer and models. There are several approaches to this step of pre-processing the images that could be explored and optimized using a MAF. However, for this effort, we are focused on the optimization of the SGD model and not the preprocessing of the data. Therefore, we are using a basic set of scaling and data reduction transformations with fixed configurations.

Training Stage

In the training stage, we configure the training module with a range of values for the initial state – which is indicated by a single integer that is used to repeatedly generate a random initial state – and the maximum number of iterations. For the initial state, we will range through values from 0-50 and, for the maximum number of iterations, we will range through the values 500-5000 by 500 iteration increments. The MAF uses this information to create 450 instances of the training module. Because model training requires more resources than normalization and data access, this module takes the form of a Docker container. This allows some resources from the host machine to be better utilized more easily. The MAF connects all 450 instances to the normalized training pathway and the normalized testing pathway. Once each module has completed its training and testing, they post the generated SGD model and its performance on the candidate model pathway.

Competition Stage

The competition stage consists of a single docker container module that is listening to the normalized competition pathway and the candidate model pathway. This module processes the candidate models one at a time as they come in. The module uses the self-reported performance on the test data and the results of running each model against the data on the competition pathway, to determine if the new model is better than the previous best model. If it is, then the model is put on the best model pathway. The updating of the best model pathway is intended to reflect a real-world scenario in which a new model is created on an interval based on batched data.

Deployment Stage

The final module in this workflow is the deployment module. This module is intended to represent a monitoring type operation where the model is deployed to perform a task. The deployment module accepts the model definition from the best model pathway and starts running data through the new model. The output from this module is simply images that were identified to have aircraft in them. This pathway could easily be picked up by another module that reports findings or even another workflow that identifies the model of the found fixed-wing aircraft or the friend/foe status of the pictured aircraft.

Outcomes

We were able to extract the code from past projects into modules with descriptions using the Module Definition artifacts. We identified the data that each module needed as input and the data that each module produced and described them using the Data Definition. Finally, we were able to fully describe the desired workflow needed to reproduce our previous efforts using the Workflow Definition.

Running this Workflow Definition in our MAF implementation allowed us to reproduce the results of our previous effort within the MAF. The MAF workflow ran in roughly the same amount of time as the monolithic code. This is because we were running on the same hardware and parallelization was limited to the number of cores on that machine. However, moving the MAF to a distributed environment would require changes to the MAF but the modules would remain the same. This separation of the modules from the deployment environment means that the development for a given deployment environment only needs to happen on the MAF side and the modules can be developed independently and tested on any MAF. In theory, the workflow we tested would be substantially faster in a distributed MAF than the monolithic implementation ever could be.

CONCLUSION AND FURTHER EFFORTS

MAFs already exist and are being used in many domains and environments. However, existing MAF implementations seem to be intended as a rapid workflow development tool. There is a great deal of potential for MAFs to go beyond a development tool and become a tool that provides flexibility in deployment, remote data analysis, and increases in performance. There is also potential to safely expose data sources that have previously been locked behind restrictive environmental constraints. One of the limitations currently preventing the application of MAFs to solve these problems is the lack of a standard way of describing the instructions for a MAF. We have shown that such standards could provide benefits in code reuse and maintenance as well as the benefits of universal deployment.

There is still a good deal of work needed to fully develop standards for MAF definitions. This includes accounting for more module use-cases as well as a better understanding of MAF deployment in different environments, namely, the features and limitations of those environments that need to be accounted for.

Several algorithms need to be refined or developed to determine the best way to handle data pathway implementation. Currently, these algorithms are crude, assume one-size-fits-all solutions, and require a good deal of potentially unnecessary processes to get data on and off the data pathways. One possible solution to this problem that should be explored is the use of shared memory between modules, allowing them to create and use data objects in more effective ways. A shared memory data infrastructure should be relatively simple on single machine implementations of MAFs and should be possible using tools like Memcached in distributed environment implementations (What is Memcached?, 2024).

Finaly, efforts could greatly improve module deployment optimization. Currently, MAFs use either prebuilt modules or limit users to only develop modules using well understood languages and tools. These limitations mean that the MAF only needs to have one way of deploying modules. With the goal of open MAFs to allow the best tool to be used for each module, and knowing the best tool may depend on the deployment environment, there is a need to allow the MAF to "make decisions" about the best way to deploy any given module in a workflow. This could mean that modules may need to be submitted to a MAF as raw code and the MAF decides the best way to compile and package that code for the known environment. This is a complicated problem but would massively reduce the amount of expertise needed to create modules and workflows allowing for a more universal adoption of MAFs and empowering users to use the most effective tools to solve their problems.

REFERENCES

Buratti, B. J., Eichmann, P., Shang, Z., Zgraggen, E., Blanc, J., Bowditch, N., . . . Yang, P. (2023, 02). Should Drag-and-Drop Analytics Become Part of the Data Scientist Toolkit? doi:10.13140/RG.2.2.27359.53921

(2021). *Framework for Access to All of Us Data Resources v1.1.* All of us Research Program. Retrieved from https://www.researchallofus.org/wp-content/themes/research-hub-wordpress-theme/media/data&tools/data-access-use/AoU_Data_Access_Framework_508.pdf

Hahmann, M., Hartmann, C., Kegel, L., Habich, D., & Lehner, W. (2016). Big by blocks: modular analytics. *it - Information Technology, 58*(4), 176-185. doi:doi:10.1515/itit-2016-0003

Jamshidi, M. (Ed.). (2008). *System of Systems Engineering: Innovations for the 21st Century* (1 ed.). Wiley.

MQTT: The Standard for IoT Messaging. (2024). Retrieved from MQTT: https://mqtt.org/

OMG Systems Modeling Language (OMG SysML™) Version 1.6. (2019). OMG Systems Modeling Language. Retrieved from SysML.org: https://sysml.org/.res/docs/specs/OMGSysML-v1.6-19-11-01.pdf

Open Images Dataset V7: Download. (2024, June). Retrieved from Open Images Dataset V7: https://storage.googleapis.com/openimages/web/download_v7.html

Platform Analytics Capabilities | Alteryx_2024. (2024, March). Retrieved from Alteryx: https://www.alteryx.com/products/capabilities

PostgreSQL: The World's Most Advvanced Open Source Relational Database. (2024). Retrieved from PostgreSQL: https://www.postgresql.org/

Software overview | KNIME. (2024). Retrieved from KNIME: https://www.knime.com/software-overview

What is Memcached? (2024). Retrieved from Memcached: https://memcached.org/